5. Rishabh

Program Name: Rishabh.java **Input File: rishabh.dat**

In researching the concept of string parity, Rishabh encountered some work done by Richard Hamming in 1950. His work produced an innovative way to improve this error-checking process to ensure data is accurately and reliably transmitted. Parity is often referred to as ODD or EVEN, by adding up the bit values in a string. If EVEN parity is used, the string 10110101, which has 5 bits, is appended with the value 1 to make the total number of bits even. If ODD parity is used, the same string would be appended with a 0, to maintain an odd total for all of the bits, resulting in 101101010 as the transmitted string.

Hamming's work produced not only a way to check IF there is an error, but WHERE the error is in the string. Supposedly, a parity code based on Hamming's technique is said to be a perfect code in reliably transmitting and fixing strings that become corrupted. Here is an example using even parity. For a bit string of length 8, which is 2^3, 4 bits can be added to create Hamming parity string. For a string of 2^N bits, the leftmost bit is considered position 1 and the rightmost bit is position 2^N. The parity bits will be placed at each position that is a power of 2 (i.e., at positions 1, 2, 4, 8, 16, …), with the actual data bits starting in position 3, then 5, 6, 7, 9, 10, ... and so on. For the string 10110101, which can be represented by the hex string B5, the Hammond parity string would start out as:

```
1 2 3 4 5 6 7 8 9 10 11 12
_ _ 1 _ 0 1 1 _ 0  1  0  1
```

To find the bit value in position 1, add up all of the odd position bits, with any blank counting as a zero. This would be 0+1+0+1+0+0, for a total of 2, which is even, therefore no bit needs to be added, making the value of position 1 zero.

```
1 2 3 4 5 6 7 8 9 10 11 12
0 _ 1 _ 0 1 1 _ 0  1  0  1
```

To find the bit value in position 2, start at position 2 and use two bits, then skip 2, use 2, skip 2, and so on resulting in summing the values in positions 2, 3, 6, 7, 10, and 11, for a sum of 0+1+1+1+1+0, or 4, again even parity resulting in zero for position 2.

```
1 2 3 4 5 6 7 8 9 10 11 12
0 0 1 _ 0 1 1 _ 0  1  0  1
```

To find the bit value in position 4, start at position 4 and use 4 bits, then skip 4, use 4, skip 4, and so on resulting in summing the values in positions 4,5,6,7, and 12, for a sum of 0+0+1+1+1, or 3, needing a 1 in position 4 to make it even parity. The value of the 8 bit would use the same pattern as the others: use 8, skip 8, and so on, summing the bits in positions 9, 10, 11, and 12, for a total of 2, even parity and zero for position 8. In general, for a parity bit at position P, where P is a power of 2, the general pattern continues in the same manner: start at position P then use P values, skip P values, and so on…

Here is the final transmitted string. The added parity bits in positions 1, 2, 4, and 8 are: **0010**.

```
1 2 3 4 5 6 7 8 9 10 11 12
0 0 1 1 0 1 1 0 0  1  0  1
```

For odd parity, the example above would result in an odd parity bit string of **1101**.

```
1 2 3 4 5 6 7 8 9 10 11 12
1 1 1 0 0 1 1 1 0  1  0  1
```

Input: Several strings in hex form, each followed by the word EVEN or ODD, to indicate the parity to use. It is guaranteed that the hex string will produce no more than 128 bits.
Output: The resulting Hammond parity string for the given hex string.

Sample Input:

```
B5 EVEN
B5 ODD
CF EVEN
F EVEN
ABC ODD
```

Sample Output:

```
0010
1101
0100
111
11101
```